KERNFORSCHUNGSANLAGE JÜLICH GmbH
Zentralinstitut für Angewandte Mathematik

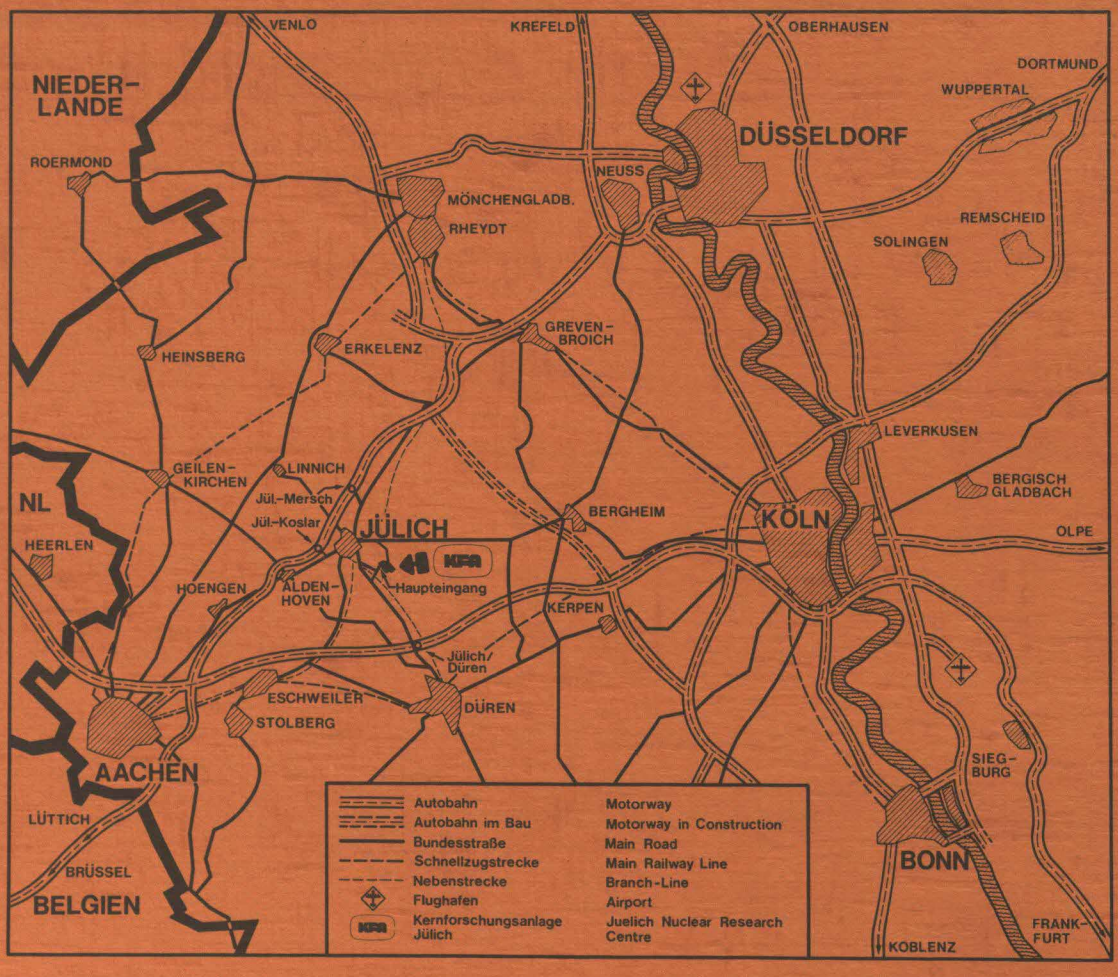
EDO

ein Text-Editor für das PDP-11 Betriebssystem RSX-11M

von

P. Basen und S. Rafflenbeul

Jül - Spez - 62
November 1979
ISSN 0343-7639



Als Manuskript gedruckt

Spezielle Berichte der Kernforschungsanlage Jülich – Nr. 62

Zentralinstitut für Angewandte Mathematik Jül - Spez - 62

Zu beziehen durch: ZENTRALBIBLIOTHEK der Kernforschungsanlage Jülich GmbH

Postfach 1913 · D-5170 Jülich (Bundesrepublik Deutschland)

Telefon: 02461/611 · Telex: 833556 kfa d

EDO

ein Text-Editor für das PDP-11 Betriebssystem RSX-11M

von

P. Basen und S. Rafflenbeul

Zusammenfassung

Für das für PDP-11 Rechner angebotene Echtzeitbetriebssystem RSX-11M ist ein Text-Editor entwickelt worden, der aufbauend auf den Editoren des Batch-Betriebssystems DOS11 und des Realzeit-Betriebssystems RT11 Benutzern eines dieser Betriebssysteme den Übergang zum RSX-11M erleichtern soll. Der neue Editor hat gegenüber seinen Vorbildern sowie dem derzeitigen RSX-11M Editor einen erweiterten Funktionssatz zur bequemeren Handhabung und zur zusätzlichen zeichenweisen Bearbeitung des Textes. Durch Ausnutzung der Speicherverwaltungseinheit (MMO) konnte bei einem virtuellen Arbeitsbereich von 20K Worten der reale Hauptspeicherbedarf für die Zeichenkette auf 6K Worte beschränkt bleiben.
Im vorliegenden Bericht sind Aufbau, Funktionsweise und Handhabung des Editors beschrieben.

Summary

Based upon the editor of the batch operating system DOS11 and the real-time operating system RT11 for PDP-11 computers, an editor running under the real-time operating system RSX-11M has been developed to minimize the effort of changing to the RSX-11M operating system. The new editor offers easier handling and character oriented manipulation of the character string by a more powerful set of commands in comparison to the editors ealier mentioned and the presently available RSX-11M editor. For the virtual working area of 20K words only 6K words of real main storage are needed by using the Memory Management Option (MMO).
This report describes the new editor, how it has been designed, how it works, and how the user has to handle it.

Inhaltverzeichnis
---

1. Einleitung

EDO ist ein Text-Editor für das Betriebssystem RSX-11M. Er wurde erstellt, um

- Benutzern, die bereits mit den Editoren der Betriebssysteme DOS11 /1/ oder RT11 /2/ gearbeitet haben, die Umstellung auf den Editor des Betriebssystems RSX-11M (RSX-Editor) /3/ zu ersparen
- erhebliche Nachteile des RSX Editors - ausschließlich zeilenweise Verarbeitung, zu kleiner Arbeitsbereich - zu beseitigen
- zusätzliche wesentliche Funktionen verfügbar zu machen.

Die Grundlage für EDO bildet der DOS-Editor, dem neue Kommandos hinzugefügt wurden und der zur besseren Handhabung eines hinreichend großen virtuellen Arbeitsbereichs die Memory Management Einheit (MM0) benutzt. Mit Hilfe des Editors können

- Text-Files erstellt bzw. modifiziert werden
- vorhandene Text-Files von einem beliebigen von RSX-11M unterstützten Speichermedium in den Arbeitsbereich gelesen werden
- Text-Files aus dem Arbeitsbereich auf ein beliebiges von RSX-11M unterstütztes Speichermedium geschrieben werden.

Ein Text-File besteht aus einer oder mehreren Seiten, die durch das Steuerzeichen FORM-FEED (FF) voneinander getrennt sind. Zur Verarbeitung werden jeweils eine oder mehrere Seiten sequentiell vom Eingabe-File in den virtuellen Arbeitsbereich gelesen und nach der Bearbeitung in den Ausgabe-File übertragen.
Im Gegensatz zum RSX-Editor gestattet EDO auch die zeichenweise Verarbeitung und verfügt über folgende auch im DOS-Editor nicht vorhandenen Funktionen:

- es können bis zu 3 Macros definiert werden
- nach jedem Suchvorgang wird die Zeile ausgegeben, auf die der Zeiger weist
- es steht ein beliebig großer Speicherbereich für das SAVE - Kommando zur Verfügung
- mit TOP OF FILE werden alle bisherigen Änderungen automatisch sichergestellt und anschließend an den Anfang des Files zurückgesprungen.
- dem Benutzer steht ein virtueller Arbeitsbereich von 20K Worten zur Verfügung.

Der Arbeitsbereich ist auf der Platte in einem automatisch angelegten und EDO zugeordneten File abgelegt. Lediglich ein Teilbereich von 6K Worten befindet sich im Hauptspeicher. Je nach Bedarf wechselt EDO den Hauptspeicherinhalt gegen ein anderes Teilstück des Arbeitsbereiches aus, ohne daß der

Benutzer hierüber Kenntnis erhält. Ausgelöst wird dieser Wechsel durch eine Funktion der Memory Management Einheit (MMO), die jeweils dann anspricht, wenn eine Adresse referiert wird, die außerhalb des Teilbereichs liegt. Die MMO - Funktion startet eine EDO - Routine, die den Wechsel durchführt.
EDO läuft als jederzeit auslagerbare (checkpointable) RSX-Task und benötigt 18K Worte Hauptspeicher.

2. Arbeitsweise des Editors

Der gesamte Editor einschießlich des Arbeitsbereiches ist in einen virtuellen Adreßbereich von 32K Worten untergebracht. Er befindet sich unter dem File-Namen `BUFF.EDI` auf der Systemplatte.
Nach Aufruf des Editors wird das Programm in den Kernspeicher geladen und ein 6K Worte großer Arbeitspuffer zur Aufnahme des oben genannten Teilbereichs bereitgestellt. Auf diese Weise ist es möglich, den Hauptspeicherbedarf in vertretbaren Grenzen zu halten.
Der zu bearbeitende Gesamttext, sofern ein solcher bereits vorliegt, befindet sich, logisch nach Seiten geordnet, im Eingabe-File (Abb. 1) auf einem beliebigen Medium.

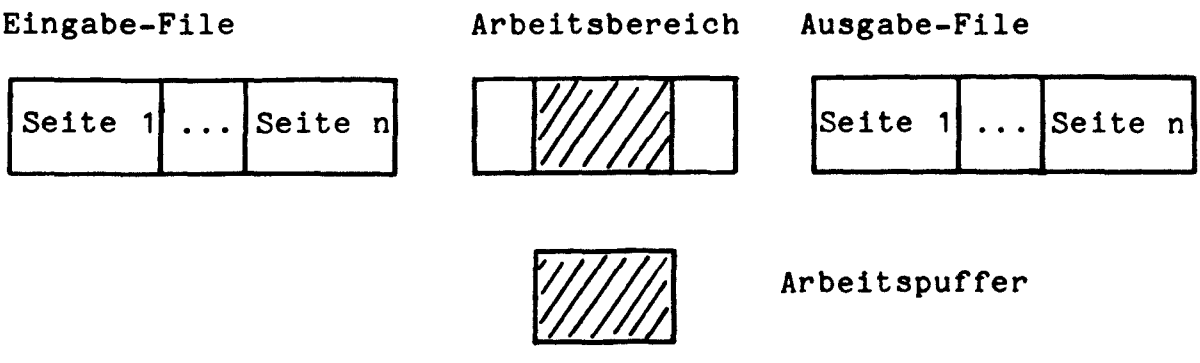


Abb. 1

Zur Bearbeitung wird eine Seite (jedoch maximal 20K Worte) aus dem Eingabe-File in den Arbeitsbereich geladen. Der jeweils zu verarbeitende Teil des Arbeitsbereiches befindet sich im Arbeitspuffer. Wird der Inhalt des Arbeitsbereiches nicht mehr benötigt, so wird er in den Ausgabe-File übertragen.
Der Austausch der Daten zwischen dem Arbeitsbereich auf der Platte und dem Arbeitspuffer im Kernspeicher erfolgt

automatisch unter Zuhilfenahme der unter dem Betriebssystem RSX-11M zur Verfügung stehenden speziellen Funktion, die sich ihrerseits wieder der Memory Management Einheit bedient, sobald eine virtuelle Adresse referiert wird, die außerhalb des Arbeitspuffers liegt. Der Arbeitspuffer wird durch ein residentes Overlay abgebildet. Dadurch erhält die Task auf einfache Weise ein Fenster zum Arbeitsbereich. Der Executive gegenüber bestehen Editor und Arbeitsbereich aus einer Task, die aber auch bei Bedarf komplett ausgelagert werden kann (checkpointing). Eine andere denkbare Lösung, nämlich zur Laufzeit eine dynamische Region zu erstellen, hätte den Nachteil, daß diese Region in einer Partition mit dem Attribut 'SYS' zur Laufzeit des Editors fest installiert würde und nur der Editor auslagerbar wäre. Die Region selber jedoch kann weder ausgelagert noch verschoben werden, was in einem Multi-User-System unzweckmäßig wäre. Das Laden des residenten Overlays in den Kernspeicher erfolgt durch den Aufruf des Unterprogramms OVL, das in dem Overlay enthalten ist und einmalig ausgeführt wird. OVL löscht den gesamten Arbeitsbereich, übergibt die virtuelle Adresse des residenten Overlays - und damit die Anfangsadresse des Arbeitspuffers - an den Editor. Das Unterprogramm OVL wird nach Ausführung von dem 6K grossen Arbeitspuffer überlagert. Der Arbeitsbereich ist logisch aufgeteilt in 8 4K-Blöcke, entsprechend den 8 Memory Management Seiten (siehe Abb.2). Zur Verarbeitung werden jeweils

1,5 Seiten = 6K Worte

von der Platte in den Arbeitspuffer gebracht. Die halbe Seite dient zur Überlappung bei Befehlen, die über die Seitengrenzen hinausgehen. Es gibt keine Befehle, die über 2 Seitengrenzen hinausgehen, da größere Befehle in Teilbefehle aufgelöst werden.

Wenn bei Ausführung eines Editor-Befehls eine Adresse angesprochen wird, die außerhalb des im Arbeitspuffer befindlichen Teils des Arbeitsbereichs liegt, wird von der Memory Management Einheit (MMO) eine Trap (interner Interrupt) generiert, die über das Macro SVTK$ an eine editor-eigene Routine weitergeleitet wird. In dieser Routine wird der alte Arbeitspuffer (1,5 Seiten) in den Arbeitsbereich zurückgeschrieben. Anschließend werden die verlangte und die folgende halbe Seite in den Arbeitspuffer gelesen. Eine Ausnahme bildet die Seite 7, d.h. die Seite am Ende des Arbeitsbereiches; es wird hier nur die letzte Seite geladen.

Wird diese Trap durch Überschreiten der unteren Grenze ausgelöst, so werden die vorhergehende Seite und die erste Hälfte der aktuellen Seite neuer Inhalt des Arbeitspuffers.

Tritt die Trap an der oberen Grenze auf, so werden die nächste Seite, deren erste Hälfte sich bereits im KSP befunden hat, und die erste Hälfte der übernächsten Seite in den KSP geladen.

Das heißt, das Fenster wird um eine Seite nach oben verschoben; es existiert immer eine Überlappung von einer halben Seite, so daß Befehle, die sich auf Text an den Grenzen des Arbeitspuffers beziehen, vollständig ausgeführt werden können.

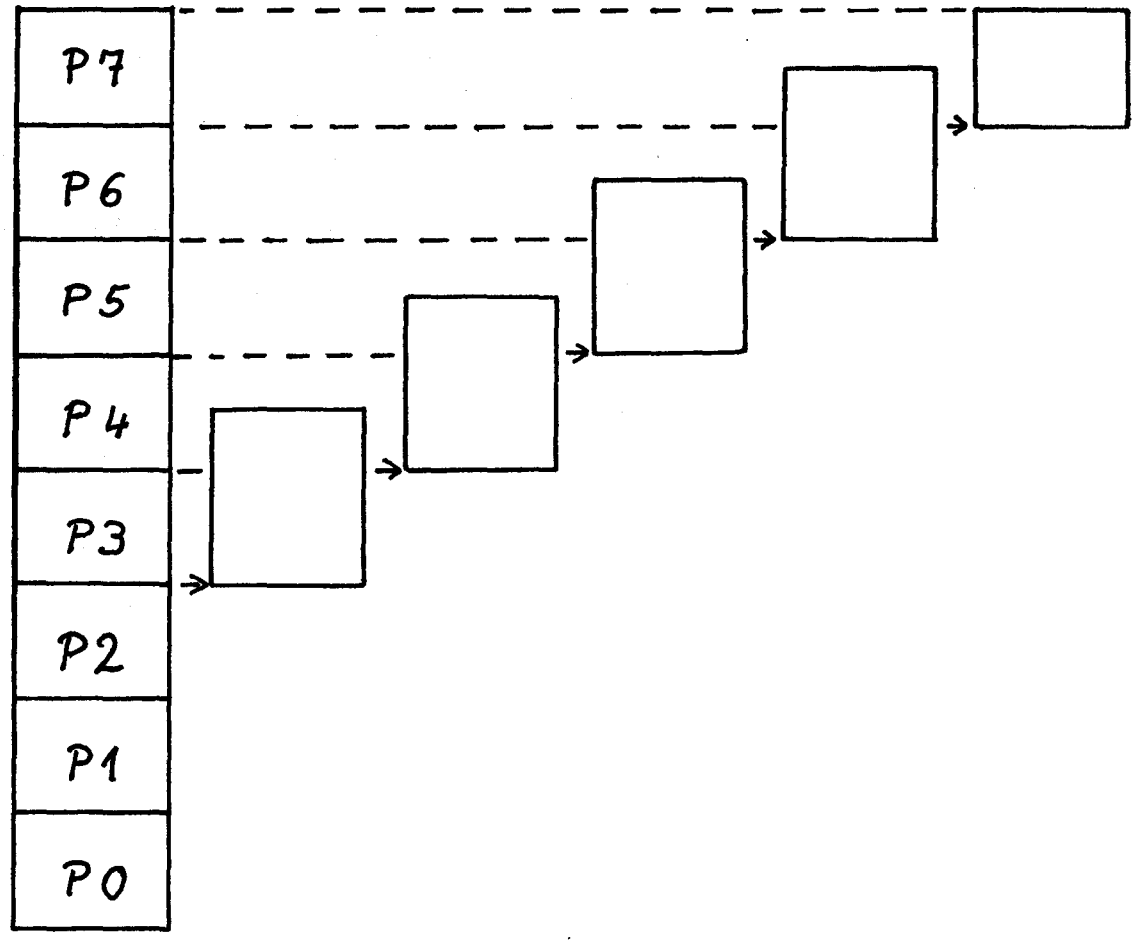


Abb. 2 Unterteilung des Arbeitsbereichs in Seiten und Darstellung der Teilbereiche, die jeweils in den Kernspeicher geladen werden.

(P0,P1 und P2 werden zur Zeit nicht benutzt.)

Bei der Initialisierung werden zuerst Ausgabe- und, falls vorhanden, Eingabe-File geöffnet. Der Zugriff auf diese Files erfolgt record-weise (GET$,PUT$), wobei die Records variable Längen haben, maximal jedoch 130 Zeichen lang sind.

Anschließend wird der Arbeitsbereich geöffnet oder, falls noch nicht vorhanden, zuerst als zusammenhängender (contiguous) File von 128 Plattenblöcken angelegt und dann geöffnet. Der Arbeitsbereich wird blockweise verarbeitet mit Blöcken zu 30000(8) Bytes, das entspricht 1,5 Seiten. Nur der letzte Block besteht aus 20000(8) Bytes, also einer Seite.
Der Arbeitsbereich wird zu Beginn einmal über die ganze Länge auf Null gesetzt. Falls ein Eingabe-File vorhanden ist, erfolgt ein automatisches Read auf die erste Textseite. Jedes Read (auch bei 'Next' und 'Top of File') liest recordweise eine Textseite, d.h. bis zum nächsten <FF> oder bis zum Erreichen einer maximalen Grenze LASTIP (virtuelle Adresse 160000) vom Eingabe-File in den Arbeitsbereich. Das bedeutet, es bleibt eine Seite frei für Texterweiterungen. Beim Einlesen wird jedem Record ein <CR><LF> angefügt, mit Ausnahme der Records, die mit <FF> enden.

Nachdem die Files geöffnet sind und eventuell die erste Seite eingelesen ist, kann die Textverarbeitung beginnen. (Editor Befehle siehe Kapitel 6)

Die Befehle 'Save' und 'Unsave' arbeiten mit einem gesonderten Save-File SAVFIL.TMP. Sobald sich ein 'Save' auf mehr als eine Zeile bezieht, wird der Save-File geöffnet oder, falls er noch nicht existiert, neu angelegt für Record-I/O mit Records einer festen Länge von 134 Bytes. Der sicherzustellende Text wird Zeile für Zeile (Record für Record) auf den File geschrieben, wobei das erste Wort jedes Records die aktuelle Länge angibt.
Der erste Record des Files enthält im ersten Wort die Gesamtanzahl der sichergestellten Zeichen und im zweiten Wort die Anzahl der sichergestellten Zeilen. So ist es jederzeit möglich, den Save-File auch bei späteren Editor-Sitzungen zu verwenden, solange er nicht explizit vom Benutzer gelöscht wird.
Jedes neue 'Save' über mehr als eine Zeile und jedes 'Unsave' können auf den Save-File zugreifen. Bei 'Exit' oder 'Top of File' wird der Save-File geschlossen.
Falls sich ein 'Save' nur auf 1 Zeile bezieht, ist hierfür ein spezieller Puffer im Programmbereich bereitgestellt.

Jedes 'Next' schreibt die aktuelle Textseite Record für Record in den Ausgabe-File, wobei <CR><LF> nicht mit übertragen werden, und liest die folgende Textseite ein. Der Befehl 'Top of File' arbeitet zunächst wie 'Exit', springt aber nicht in den Monitor zurück. Statt dessen wird der Ausgabe-File zum neuen Eingabe-File, und es wird ein neuer Ausgabe-File mit gleichem Namen eröffnet und die erste Seite eingelesen.

Die Editor-Verarbeitung wird mit einem 'Exit' abgeschlossen; 'Exit' schreibt die aktuelle Textseite in den Ausgabe-File (s. 'Next'), liest alle folgenden Seiten vom Eingabe-File und schreibt sie sofort wieder in den Ausgabe-File, bis das Ende des Eingabe-Files erreicht ist. Dann werden alle geöffneten Files geschlossen. Sollte der Ausgabe-File leer sein, wird er anschließend gelöscht. Über die EXIT$-Directive erfolgt der Rücksprung in den Monitor.

3. Kernspeicherbelegung

Der Adressbereich 0-60000(8)der Task wird vom Programm belegt. Die nächsten zusammenhängenden 30000(8)Bytes Adressraum sind für den Arbeitspuffer (Overlay) reserviert.

4. File-Strukturen

Zur Arbeit mit dem Editor wird eine Anzahl von Files benötigt. Hier erfolgt eine genauere Beschreibung ihrer Eigenschaften.

4.1 Arbeitsbereich (BUFF.EDI)

Der Arbeitsbereich wird angelegt als zusammenhängender (contiguous) File von 128 physikalischen Blöcken, wobei ein Block 512 Bytes entspricht.
Das Öffnen geschieht mit OPEN$M, falls der File schon existiert, anderenfalls mit OPEN$W.

a) Block I/O	FD.RWM
b) Blockgröße	30000(8) bzw. 20000(8) Bytes
c) Open for Modification	FO.MFY
d) System-Device 0	SY0:

4.2 Eingabe-File (in.MAC)

a) Record-I/O mit variabler Länge	R.VAR
b) maximale Record-Größe	130 Bytes
c) Format: <LF>record<CR>	FD.CR
d) Open for Reading only	FO.RD
e) System-Device 0 (default)	SY0:

Der Eingabe-File wird mit OPEN$R geöffnet.

4.3 Ausgabe-File (out.MAC)

a) Record-I/O mit variabler Länge	R.VAR
b) maximale Record-Größe	130 Bytes
c) Format: <LF>record<CR>	FD.CR
d) Open for Writing, Create new File	FO.WRT
e) System Device 0 (default)	SY0:

Der Ausgabe-File wird mit OPEN$W geöffnet.

4.4 Save-File (SAVFIL.TMP)

a) Record-I/O mit fester Länge	R.FIX
b) Recordgröße	134 Bytes
c) Format: Die Records werden übernommen, wie sie ankommen	
d) Random Access	
e) Open for Modification	FO.MFY
f) System-Device 0	SY0:

Der Save-File wird entweder mit OPEN$W, falls noch nicht vorhanden, oder anderenfalls mit OPEN$U geöffnet.

5. Aufruf und Arbeitsmodi des Editors sowie Hilfsmittel

5.1 Aufruf

Wenn der Monitor durch Drucken von '>' seine Bereitschaft gezeigt hat, Kommandos anzunehmen, erfolgt der Editor-Aufruf durch Eingabe von >RUN EDO

Der Editor meldet sich mit EDITOR - V01

und in der nächsten Zeile EDO>

Hier erwartet er eine Kommandozeile der Form:

[dev:] outputfile= [dev:] inputfile

Wenn die Kommandozeile korrekt war, zeigt der Editor seine Bereitschaft zur Annahme von Editor-Befehlen, indem er ein '*' ausdruckt.

Mögliche Kommandozeilen
---

1. DEV:NAME1.EXT;VERS =DEV:NAME2.EXT;VERS
2. DEV:NAME.EXT;VERS
   a) Falls der spezifizierte File existiert, wird dieser als Eingabe-File genommen. Für den Ausgabe-File gelten die gleichen Angaben, die Versionsnummer wird um 1 erhöht. Eingabe- und Ausgabe-File werden geöffnet, und es wird automatisch ein ´Read´ auf die erste Seite ausgeführt.
   Der Editor meldet sich mit
   ```
   1ST PAGE READ IN
   *
   ```
   b) Falls der spezifizierte File nicht existiert, wird er als neuer Ausgabe-File angelegt. Der Editor meldet sich mit ´*´.

Die Angaben DEV:, EXT und VERS sind wahlweise; falls eine Angabe fehlt, gilt Folgendes:

für	DEV:	SY0: User System Device
für	EXT	MAC Macro Assembler Source
für	VERS	höchste vorhandene Versionsnummer +1

Also aus "ABC" wird "SY0:ABC.MAC;1", falls noch keine Version des Files vorhanden war.

5.2 Modi

Der Editor unterscheidet 2 Arbeitsweisen:

a/ Kommando-Modus
b/ Eingabe-Modus

Im Kommando-Modus (das Terminal meldet sich mit ´*´) erwartet der Editor ein legales Kommando, das nach <CR> ausgeführt wird. Im Eingabe-Modus wird jedes eingegebene Zeichen als Text interpretiert und verarbeitet. Der Eingabe-Modus wird erzeugt durch Kommandos, die Zeilen entweder in den Text einfügen oder gegen vorhandene Zeilen austauschen. In den Eingabe-Modus gelangt man durch das entsprechende Kommando mit anschließendem <CR>. Es wird kein ´*´ mehr gedruckt, sondern der Editor erwartet sofort die Eingabe. Wenn diese abgeschlossen ist, schaltet man durch Eingabe von <LF> mit folgendem <CR> zurück in den Kommando-Modus.

5.3 Dot

Der Dot ist ein aktueller Positionsanzeiger. Er steht im Arbeitsbereich an der Stelle, auf die sich der oder die folgenden Befehle beziehen. Bei einem speziellen Druckbefehl wird zur Verdeutlichung an der Stelle des Dots ein '^' ausgedruckt. Der Dot wird mittels spezieller Befehle - explizit oder implizit - innerhalb des Arbeitsbereiches verschoben.

5.4 Mark

Mark ist ein weiterer Positionsanzeiger, der durch einen speziellen Befehl auf die aktuelle Position des Dot gesetzt wird. Er kann von einigen Befehlen mittels des Argumentes '@'(siehe Abschnitt 'Argumente') direkt angesprochen werden. Zu Beginn des Programms zeigt Mark auf den Anfang des Arbeitsbereiches.

6. Spezielle Gruppen von Editor-Befehlen

Die Editor-Befehle können in 3 Klassen gegliedert werden:

a/ zeichenorientiert
b/ zeilenorientiert
c/ seitenorientiert

Das Argument n bedeutet

bei zeichenorientierten Befehlen die Anzahl der zu verarbeitenden Zeichen. Steuerzeichen wie <CR>, <LF> und <FF> werden jeweils als ein Zeichen behandelt

bei zeilenorientierten Befehlen die Anzahl der zu verarbeitenden Zeilen. Eine Zeile wird abgeschlossen durch <CR><LF>. Sie besteht aus maximal 132 Zeichen

bei seitenorientierten Befehlen die Anzahl der zu verarbeitenden Seiten.

7. Argumente

Argumente werden den Editor-Kommandos vorangestellt und geben an, wie oft oder innerhalb welcher Grenzen ein Befehl ausgeführt werden soll. Ausnahme bilden die Textargumente. Sie stehen hinter den entsprechenden Kommandos und sind durch 2 gleiche Trennungszeichen begrenzt.

Die folgenden Argumente bedeuten:

n	Ganze Zahl zwischen 1 und 32767; Vorzeichen '+' und '-' sind möglich: '+' vom Dot vorwärts in Richtung Bereichs-Ende '-' vom Dot rückwärts in Richtung Bereichs-Anfang
0	vom Anfang der Zeile bis zum Dot
/	vom Dot bis zum Bereichs-Ende
@	vom Dot bis zum Mark
/text/	Textargument, steht hinter dem betreffenden Befehl, wobei // Trennnungszeichen sind, die den Text von den Befehlen abgrenzen und nicht innerhalb des Textes vorkommen dürfen, text der zu verarbeitende Text ist.

Es sind nicht alle Argumente bei allen Kommandos zulässig (siehe Kommando-Aufstellung). Falls ein Argument nicht explizit angegeben ist, wird n=1 angenommen. Falls ein '-' angegeben ist, wird n=-1 angenommen.

8. Kommando-Beschreibung

Editor-Kommandos bestehen aus ein oder zwei Buchstaben und einem möglichen Argument.
Es können beliebig viele Kommandos in einer Zeile aufeinanderfolgen; die Zeile darf jedoch nicht mehr als 132 Zeichen enthalten. Abgeschlossen wird eine Kommando-Zeile durch Drücken der Return-Taste (-> <CR>), was der Eingabe von <CR><LF> entspricht. Erst mit <CR> wird die Ausführung der Befehle gestartet.

8.1 Ein-/Ausgabe- Befehle

Die I/O-Befehle dienen zur Kommmunikation zwischen I/O-Gerät und Arbeitsbereich.

R Read — liest eine Seite vom Eingabe-File in den Arbeitsbereich. Dot und Mark stehen am Anfang des Bereichs.

! END OF FILE ! besagt, daß der Eingabe-File vollständig eingelesen wurde.

N Next — schreibt den Inhalt des Arbeitsbereiches in den Ausgabe-File und liest die folgende Seite vom Eingabe-File. Dot und Mark stehen am Anfang des Bereichs.

(+)nN — schreibt und liest n Seiten

W Write — schreibt die aktuelle Zeile in den Ausgabe-File.

(+)nW — schreibt die aktuelle Zeile und n-1 folgende in den Ausgabe-File

/W — schreibt, ausgehend vom Dot, den Rest der Seite in den Ausgabe-File.

F Form Feed — schreibt ein Form Feed <FF> in den Ausgabe-File.

EX Exit — schreibt den Inhalt des Arbeitsbereiches und alle restlichen Seiten des Eingabe-Files in den Ausgabe-File, schließt alle geöffneten Files und kehrt anschließend vom Editor in den Monitor zurück.

T Top of File — macht ein Exit, jedoch ohne Rückkehr in den Monitor; statt dessen nimmt es den gerade bearbeiteten Ausgabe-File als Eingabe-File und legt als neuen Ausgabe-File einen File gleichen Namens mit einer um 1 erhöhten Versionsnummer an.
Es wird die Meldung
´TOP OF FILE´
ausgedruckt, anschließend die erste Seite eingelesen mit der Meldung
´1ST PAGE READ IN´.
Dot und Mark stehen dann am Anfang des Arbeitsbereiches.

8.2 Kennzeichnen einer Stelle

M Mark — setzt einen Zeiger (Mark) an die aktuelle Position des Dot. Mit dem Argument '@' kann diese Position angesprochen werden.

8.3 Druckbefehle

L List — druckt, ausgehend vom Dot, Zeilen auf dem Terminal aus.

(+)nL	druckt n Zeilen hinter dem Dot
-nL	druckt n Zeilen vor dem Dot
/L	druckt Text vom Dot bis zum Ende des Arbeitsbereiches
0L	druckt aktuelle Zeile vom Beginn bis zum Dot
@L	druckt Zeilen vom Dot bis zum Mark

V Verify — druckt die aktuelle Zeile und - abhängig von VERFA - eine gewisse Anzahl von Zeilen vor und hinter der aktuellen Zeile.

VERFA=	0	V druckt nur aktuelle Zeile
	(+)n	V druckt aktuelle Zeile und n Zeilen vor und hinter ihr
	<0	V hat keine Wirkung

<CR> — Wird in einer Kommando-Zeile nur die Return Taste gedrückt, wird ein 'V' ausgeführt.

Q Qualify — weist VERFA einen Wert zu:

0Q	->	VERFA= 0
(+)nQ	->	VERFA= n
-nQ	->	VERFA= -n

8.4 Verschieben des Dot

B Begin — setzt den Dot an den Anfang des Arbeitsbereiches.

A Advance — verschiebt den Dot um eine Anzahl von Zeilen und setzt ihn an den Anfang der neuen Zeile.

(+)nA	setzt Dot n Zeilen vor
-nA	setzt Dot n Zeilen zurück
/A	setzt Dot an das Ende des Arbeitsbereiches
@A	setzt Dot nach Mark
0A	setzt Dot an den Anfang der aktuellen Zeile

J Jump — verschiebt den Dot um eine Anzahl von Zeichen.

(+)nJ	setzt Dot n Zeichen vor
-nJ	setzt Dot n Zeichen zurück
/J	setzt Dot an das Ende des Arbeitsbereiches
@J	setzt Dot nach Mark
0J	setzt Dot an den Anfang der aktuellen Zeile

Z Z-jump — setzt Dot an das Ende der aktuellen Zeile.

8.5 Suchen einer Zeichenkette

G Get/text/ — durchsucht, ausgehend vom Dot, den Arbeitsbereich nach dem zwischen den Begrenzungszeichen stehenden Text.
Wenn 'text' gefunden ist, steht der Dot hinter dem letzten Zeichen von 'text'.
Wenn 'text' nicht vorhanden ist, steht der Dot am Ende des Arbeitsbereiches, und es wird eine entsprechende Meldung ausgedruckt.

(+)nG/text/	führt Suchbefehl n-mal aus
0G/text/	entspricht 1G/text/

H wHole/text/ — durchsucht, ausgehend vom Dot, den Arbeitsbereich nach dem Zwischen den Begrenzungszeichen stehenden Text.

Wenn 'text' gefunden ist, steht der Dot hinter dem letzten Zeichen von 'text'.
Wenn 'text' nicht vorhanden ist, wird der Inhalt des Arbeitsbereiches in den Ausgabe-File geschrieben und die nächste Editor-Seite eingelesen und durchsucht.
Falls 'text' im Eingabe-File nicht existiert, steht der Dot am Anfang des leeren Arbeitsbereiches, und alle Editor-Seiten sind in den Ausgabe-File kopiert.
Mark bleibt unverändert, solange keine neue Editor-Seite eingelesen wird. In diesem Fall steht Mark am Bereichsanfang.

(+)nH/text/	führt Suchbefehl n-mal aus
OH/text/	sucht das nächste Auftreten von Text, aber kopiert die dazwischenliegenden Seiten nicht in den Ausgabe-File.

P Position — schreibt den aktuellen Inhalt des Arbeitsbereiches in den Ausgabe-File, liest die nächste Editor-Seite ein und durchsucht diese nach dem angegebenen Textstring.

Format: P<CR>
text<LF><CR>

Wenn 'text' gefunden ist, steht der Dot hinter dem letzten Zeichen von 'text'.
Wenn 'text' im aktuellen Arbeitsbereich nicht vorhanden ist, wird die nächste Editor-Seite eingelesen und weitergesucht. Der alte Inhalt wird dabei überschrieben, ohne in den Ausgabe-File kopiert worden zu sein.
Wenn 'text' im Eingabe-File nicht existiert, steht der Dot am Anfang des leeren Arbeitsbereiches. Die Editor-Seiten zwischen dem Arbeitsbereich zur Zeit des Kommandos und dem aktuellen Inhalt mit der gesuchten Zeichenkette bzw. dem File-Ende sind verloren.
Mark bleibt unverändert, solange keine neue Editor-Seite eingelesen wird. In diesem Fall steht Mark am Bereichsanfang.

8.6 Textverarbeitung

K Kill

löscht eine Anzahl von Zeilen im Arbeitsbereich, beginnend beim Dot. Der Dot wird nicht verändert. Mark bleibt unverändert; falls Mark jedoch im zu löschenden Bereich liegt, wird er auf den Bereichsanfang gesetzt.

(+)nK	löscht n Zeilen hinter dem Dot
-nK	löscht n Zeilen vor dem Dot
/K	löscht Bereich vom Dot bis zum Ende
@K	löscht den Bereich von Dot bis Mark
0K	löscht Zeile vom Anfang bis zum Dot

D Delete

löscht eine Anzahl von Zeichen ausgehend vom Dot. Der Dot wird nicht verändert. Mark bleibt unverändert; falls Mark jedoch im zu löschenden Bereich liegt, wird er auf den Bereichsanfang gesetzt.

(+)nD	löscht n Zeichen hinter dem Dot
-nD	löscht n Zeichen vor dem Dot
/D	löscht Bereich vom Dot bis zum Ende
@D	löscht den Bereich von Dot bis Mark
0D	löscht die Zeichen vom Zeilenanfang bis zum Dot

I Insert

fügt
a) Zeichenketten
b) beliebig viele Textzeilen
in den Arbeitsbereich ein. Der Dot steht anschließend hinter dem eingefügten Text. Mark bleibt unverändert.

a) I/text/ Insert mit Textargument
fügt 'text' vor dem Dot in die Zeile ein.

b) I<CR> Umschalten in Eingabe-Modus
fügt eine oder mehrere Zeilen vor dem Dot ein; der Rücksprung in den Kommando-Modus erfolgt durch Eingabe von <LF> und anschließendem <CR>.

X eXchange

löscht eine Anzahl Zeilen und fügt an der betreffenden Stelle neue ein. Der Dot zeigt anschließend auf die erste Zeile hinter dem eingefügten Text. Mark bleibt unverändert; falls Mark jedoch im zu löschenden Bereich liegt, wird er auf den Bereichsanfang gesetzt. X ist eine Kombination aus den Befehlen K und I.

(+)nX	löscht n Zeilen hinter dem Dot und schaltet in den Eingabe-Modus
-nX	löscht n Zeilen vor dem Dot und schaltet in den Eingabe-Modus
/X	löscht den Bereich vom Dot bis zum Ende und schaltet in den Eingabe-Modus
@X	löscht den Bereich von Dot bis Mark und schaltet in den Eingabe-Modus
0X	löscht die Zeile vom Anfang bis Dot und schaltet in den Eingabe-Modus

C Change /text/ — löscht eine Anzahl Zeichen im Arbeitsbereich und fügt 'text' an der betreffenden Stelle ein. Der Dot steht anschließend hinter dem eingefügten Text. Mark bleibt unverändert; falls Mark jedoch im zu löschenden Bereich liegt, wird er auf den Bereichsanfang gesetzt. C ist eine Kombination aus den Befehlen D und I.

(+)nC/text/	löscht n Zeichen hinter dem Dot und fügt 'text' dort ein
-nC/text/	löscht n Zeichen vor dem Dot und fügt 'text' dort ein
/C/text/	löscht den Bereich vom Dot bis zum Ende und fügt 'text' dort ein
@C/text/	löscht den Bereich von Dot bis Mark und fügt 'text' dort ein
0C/text/	löscht die Zeichen vom Anfang der Zeile bis zum Dot und fügt 'text' dort ein
=C/text/	ersetzt die beim letzten Suchbefehl gefundene Zeichenkette durch 'text'

S Save — kopiert eine Anzahl von Zeilen, ausgehend vom Dot, in einen Sicherstellungsbereich. Der alte Inhalt wird überschrieben. Der Arbeitsbereich wird nicht verändert, die Positionen von Dot und Mark bleiben erhalten.

(+)nS kopiert n Zeilen in den Sicherstellungsbereich.

U Unsave — fügt den Inhalt des Sicherstellungsbereiches vor dem Dot in den Arbeitsbereich ein. Der Dot steht hinter dem eingefügten Text, Mark ist unverändert. Der Sicherstellungsbereich bleibt erhalten.

8.7 Editor-Macros

m0 Define Macro — füllt den m-ten Macro-Puffer mit der Zeile, auf die der Dot zeigt. Es gibt 3 Macro-Puffer. Die Position des Dot und der Inhalt des Arbeitsbereiches werden nicht verändert.

10	füllt den ersten Puffer
20	füllt den zweiten Puffer
30	füllt den dritten Puffer

EMm Execute Macro — nimmt den Inhalt des m-ten Macro-Puffers als Befehlszeile und führt die Befehle aus.

(+)nEMm führt den Inhalt des Macro-Puffers m n-mal aus

8.8 Wiederholungskommando

(+)n<befehlsfolge> — die Befehlsfolge in den spitzen Klammern wird n-mal ausgeführt.

Tabelle

Nachstehend folgt eine alphabetische Aufstellung sämtlicher Befehle und ihrer Auswirkungen auf Dot und Mark (Bereich - Arbeitsbereich).

Befehl	Dot	Mark
A dvance	n Zeilen vor, zurück	-
B egin	auf Bereichsanfang	-
C hange	hinter geänd. Text	-
D elete	-	-
EM exec.macro	-	-
EX it	-	-
F orm feed	-	-
G et	hinter ges. String	-
H whole	hinter ges. String	auf Bereichsanfang
I nsert	hinter eingef. Text	-
J ump	n Zeichen vor, zurück	-
K ill	-	-
L ist	-	-
M ark	-	auf Dot
N ext	auf Bereichsanfang	auf Bereichsanfang
O def.macro	-	-
P osition	hinter ges. String	auf Bereichsanfang
Q ualify	-	-
R ead	auf Bereichsanfang	auf Bereichsanfang
S ave	-	-
T op of file	auf Bereichsanfang	auf Bereichsanfang
U nsave	hinter eingef. Text	-
V erify	-	-
W rite	-	-
X exchange	hinter geänd. Text	-
Z -jump	am Ende der akt. Zeile	-

9. Sonderzeichen

Sonderzeichen, wie Control Characters und <Rubout>, werden nicht vom Editor verarbeitet, sondern werden vom Terminal Driver abgehandelt.

Ausnahme bildet <CTRL/C>.

Damit <CTRL/C> nicht wie üblich die Monitor Control Routine (MCR) aufruft und so die Möglichkeit besteht, außerhalb der Editor-Kontrolle zu geraten, wird <CTRL/C> mit Hilfe eines 'Asynchronous System Traps abgefangen und vom Editor selbst

verarbeitet. Wenn <CTRL/C> gegeben wurde, macht der Editor ein Exit, d.h. er springt in die Exit-Routine, als ob ein 'EX'-Befehl gegeben worden wäre.

10. Macros

Der Editor bietet die Möglichkeit, bis zu 3 Kommandofolgen abzuspeichern und über einen bestimmten Befehl jederzeit während einer Editor-sitzung wieder aufzurufen. Für diese Befehlsfolgen (Macros) stehen 3 Puffer zu je 132 Bytes zur Verfügung: MAC1, MAC2, MAC3.

Puffer MAC1 wird gefüllt mit dem Befehl '10'. Das bedeutet, '10' kopiert die betreffende Zeile, auf die der Dot zeigt, in den Puffer MAC1. Der Dot bleibt unverändert. Für MAC2 und MAC3 gelten entsprechend '20' und '30'.

Mit dem Befehl 'nEM1' ('nEM') wird der Inhalt von MAC1 als eingegebene Kommandozeile interpretiert und n-mal ausgeführt. Bei 'EM2' wird entsprechend der Inhalt von MAC2 genommen, bei 'EM3' der Inhalt von MAC3. Wenn der betreffende Puffer leer sein solte, erfolgt eine Fehlermitteilung.

Die Pufferinhalte von MAC1, MAC2 und MAC3 können jederzeit durch erneute Eingabe von '10', '20' oder '30' überschrieben werden. Die Puffer werden bei 'Top of File' nicht gelöscht.

11. Wiederholungskommando

Auch ohne Macro-Benutzung besteht die Möglichkeit, eine soeben eingegebene beliebige Befehlsfolge mehrmals auszuführen. Hierzu wird die Befehlsfolge in spitze Klammern <> eingeschlossen. Das Argument n vor dem Klammerausdruck gibt an, wie oft die Folge wiederholt werden soll. Es ist erlaubt, bis zu 10 Befehlsfolgen ineinander zu verschachteln.

Beispiel: 5<1A2<2J1D>V>

In 5 aufeinanderfolgenden Zeilen wird 2 mal jedes 3. Zeichen gelöscht; anschließend wird die Zeile aufgelistet.

Die Eingabe von <CR> innerhalb einer Befehlsfolge unterbricht diese und erzeugt eine Fehlermitteilung.

12. Fehlermeldungen - Mitteilungen

Der Editor liefert eine Anzahl von Mitteilungen und Fehlermeldungen. Die Art des jeweiligen Fehlers ist aus der Meldung zu ersehen. Im folgenden werden sämtliche Mitteilungen, ihr Auftreten und der weitere Programmablauf alphabetisch geordnet aufgeführt.

BUFFER OVERFLOW	kein Platz mehr im Sicherstellungs-, Command- oder Insert-Bereich -> nächster Befehl
CLOSE ERROR	Fehler beim schließen des Ausgabe-, oder Eingabe-Files bzw. des Sicherstellungs- oder Arbeitsbereichs -> listet FDBs + Exit
DEVICE NOT ALLOCATED-MOUNTED	System kennt Zusatz-Device nicht -> Exit
EDIT BUFFER ALMOST FULL	nach ausgeführtem 'Unsave' ist im Arbeitsbereich nur noch für maximal 131 Zeichen Platz
EDIT BUFFER NEARLY FULL	bei Ausführung von 'Read'; kein Platz mehr im Arbeitsbereich zum (vollständigen) Einlesen einer weiteren Textseite; die letzte Seite im Arbeitsbereich wird für 'Insert' etc. freigehalten Abhilfe: bei Bedarf den Rest der unvollständigen Textseite mit 'Next' in den Arbeitsspeicher holen
EDITOR - V01	Angabe der Editor-Version beim Start
EMPTY MACRO BUFFER	Befehl 'EM' auf einen leeren Puffer -> nächster Befehl
EMPTY SAVE BUFFER	Befehl 'U' auf einen leeren Sicherstellungsbereich -> nächster Befehl

?EM NOT ALLOWED IN ITERATION BRACKETS?	Fehler bei 'Execute Macro' -> nächster Befehl
!END OF FILE!	die letzte Seite des Editor-Files wurde eingelesen
ERROR ON CREATE WINDOW	Fehler beim Anlegen des Fensters -> listet WDB + Exit
ERROR ON ELIMINATE WINDOW	Fehler beim Löschen des Fensters -> listet WDB + Exit
ERROR ON RETURN OF AST	Fehler beim Rücksprung aus der AST-Routine -> Exit
<EXIT>	bei Ausführung des Befehls 'Exit'
GET-PUT ERROR	Fehler beim Record-I/O -> listet FDBs + Exit
ILLEGAL ARGUMENT	Befehl mit ungültigem Argument -> nächster Befehl
ILLEGAL COMMAND	ungültiger Befehl -> nächster Befehl
ILLEGAL LINE FEED	<LF> im Kommando-Modus ohne vorhergehendes <CR> -> nächster Befehl
ILLEGAL MACRO NESTING	Befehl 'EM' in einem Macro -> nächster Befehl
(INSERTED, BUT) CR,LF BEFORE DELIMITER	in Befehlen mit Textargument wurde das 2. Begrenzungszeichen vergesen; bei 'Get', 'wHole', 'Position' -> nächster Befehl; bei 'Insert' -> Text wird eingefügt, Fehlermeldung nur als Warnung bei 'Change', 'eXchange' -> Text wird eingefügt, ohne den alten Text zu löschen
INVALID COMMAND STRING	falsche File-Angabe -> neue Eingabe verlangt
MACRO OVERFLOW	Macro Puffer zu klein, eingegebene Zeile größer als 132 Zeichen -> nächster Befehl

NO INPUT FILE	Fehler bei 'Next', 'Get', 'Read' -> nächster Befehl
NO ROOM TO INSERT	Fehler bei 'Insert', im Arbeitsbereich ist nicht mehr genügend Platz für eine vollständige Zeile -> nächster Befehl Abhilfe: an geeigneter Stelle <FF> einfügen und 'Top of File' geben
NO ROOM TO UNSAVE	Fehler bei 'Unsave', der Arbeitsbereich ist zu klein, den sichergestellten Text aufzunehmen -> nächster Befehl Abhilfe: an geeigneter Stelle <FF> einfügen und 'Top of file' geben; dann 'U' erneut probieren
OPEN ERROR	Fehler beim Öffnen des Ausgabe- oder Eingabe-Files bzw. des Arbeitsbereichs -> listet FDBs + Exit
READ-WRITE ERROR	Fehler beim Block-I/O -> listet FDB + Exit
SEARCH FAILURE	die bei 'Position', 'Get' oder 'wHole' gesuchte Zeichenkette ist im Text nicht vorhanden -> nächster Befehl
SPECIFIED INPUT FILE NOT EXISTENT	in Kommandofolge angegebener Eingabe-File existiert nicht -> Exit
TOO MANY ITERATION BRACKETS	beim Wiederholungskommando sind maximal 10 Schachtelungen erlaubt -> nächster Befehl
TOP OF FILE	nach Ausführung des Befehls 'Top of file'
1ST PAGE READ IN	nach öffnen des Eingabe-Files und dem 1. automatischen 'Read'

> WITHOUT MATCHING <

beim Wiederholungskommando stimmt die Zahl der linken nicht mit der Zahl der rechten eckigen Klammern überein.
-> nächster Befehl

13. Literaturhinweise

/1/ THE DOS/BATCH HANDBOOK
DEC-11-ODBHA-A-D April 1974

/2/ RT11 SYSTEM USER´S GUIDE
Order No. DEC-11-ORGDA-A-D 1977

/3/ RSX-11M UTILITIES MANUAL
Order No. AA-5567B-TC Dezember 1977